An Evaluation of Traceability Approaches to Support Software Evolution

Siti Rochimah, Wan M. N. Wan Kadir, Abdul H. Abdullah
Software Engineering Department,
Faculty of Computer Science and Information System,
Universiti Teknologi Malaysia
siti@its-sby.edu, wnasir@utm.my, hanan@utm.my

Abstract

Requirements traceability is becoming increasingly significant element in software engineering. It provides critical function in the development and maintenance of a software system. From the software evolution point of view, requirements traceability plays an important role in facilitating software evolution. Since the evolution is inevitable, a traceability approach must take as much as possible the important influencing aspects into account to the evolution processes in order to minimize the evolution efforts. This paper evaluates several recent traceability approaches published in literature with the focus on their contributions to software evolution. The evaluation results may be used as a basis for improving requirements traceability approaches that may simplify the software evolution tasks.

1. Introduction

Requirement traceability is defined as "*the ability to describe and follow the life of a requirement, in both a forwards and backwards direction*" [1]. For over three decades, researchers keep improving requirement traceability (or 'traceability' for short) approaches to support many software engineering activities.

There are various traceability approaches that have been proposed from a very simple way, i.e. using spreadsheet, until the latest ones that apply some formal or complex techniques. Most of them using their own traceability technique specific for their related approaches.

This paper aims at evaluating several recent software traceability approaches that have been published in literature. The evaluation focus is primarily on the capability of the approaches in supporting software evolution. The initial results obtained by this evaluation can be used to indicate to what extends each approach has a capability to support software evolution. Consequently, the results can be used as a basis for improving the current approaches related to their support for software evolution. In addition, the evaluation results may also outline the desired criteria for a more holistic approach in requirements traceability.

This paper is organized as follows: Section 2 provide a brief description on the state-of-the-art software traceability approachesy. Section 3 presents the evaluation framework that is utilized to evaluate the approaches. Section 4 discusses the evaluation results as well as the rationale behind them. Section 5 presents the discussion from overall results, while Section 6 explains threat of validity relating the evaluation results. Finally, Section 7 presents the conclusion.

2. Overview of Traceability Approaches

We reviewed about hundred of recent papers relating traceability topics. Based on them, we interested in seven approaches that include specific subject, i.e. they put the requirement as one of the main artifacts to perform traceability.

We resume here the traceability approaches, which will be further evaluated. We cite each approach in term of the mechanisms and algorithms that are utilized and scope of tracing that is covered. Those characteristics will be used to evaluate the approach in the next sections.

2.1 Information Retrieval Approach (IRA)

Recently, there are many researchers attempting to establish traceability link via information retrieval approach [2-9]. This approach focuses on automating the generation of traceability link by similarity comparison between two types of artifacts. The two basic IR models which commonly used in traceability generation are probabilistic and vector space models. Numerous variant models have also been applied including the popular model Latent Semantic Indexing which is based on vector space model. In each model, one type of particular artifacts treats as a query and another type of artifacts treats as a document being searched in term of the query. For example, source code treats as a query against requirements specification as a document being searched based on the query.

The general steps include (i) preprocessing, i.e. stop-word removal and or stemming, (ii) analyzing and indexing of an incoming document collection, followed by constructing a representation of each document and then archiving them, (iii) analyzing and representing an

International Conference on Software Engineering Advances(ICSEA 2007)
0-7695-2937-2/07 $25.00 © 2007 IEEE

incoming queries, and using a matching or ranking algorithm to determine which document representations are similar to the query representation. The scope of tracing covers almost of artifacts including high-level and low-level requirements, manual documents, design elements, test cases, and source code.

2.2 Rule-Based Approach (RB)

Spanoudakis et al. in [10, 11] propose a method to automatically create traceability link using rules. They use two traceability rules, i.e. requirement-to-object-model traceability rule (RTOM rule) and inter-requirement traceability rule (IREQ rule). The rules are deployed into three specific documents types, i.e. (i) requirements statement documents (RSD), (ii) use case documents (UCD), and (iii) analysis object models (AOM). RTOM rules are used to trace the RSD and UCD to an AOM, while IREQ rules are used to trace between RSD and the UCD. The method assumes that all of document types are in XML-based format. The traceability rules are also represented in XML-based markup language.

The method consists of four stages, i.e. (i) grammatical tagging of the artifacts, (ii) converting the tagged artifacts into XML representations (iii) generating traceability relations between artifacts, and (iv) generating traceability relations between different parts of the artifacts. RB covers requirement statement documents, use case documents, and analysis object models as the objects of tracing.

Also, Nentwich et al. in [12] build rule-based tool for consistency management for XML-based software artifacts, namely 'xlinkit'. First-order logics are utilized to describe consistency rules. The tool can be applied to all kind of textual software artifacts as long as these artifacts are in document-object-model (DOM) trees format

2.3 Event-Based Approach (EB)

Cleland-Huang et al. in [13-15] propose event-based approach for updating and maintaining traceability relationships. Traceability relationships are defined as publisher-subscriber relationships in which dependent artifacts must subscribe to the requirements on which they depend. When a requirement change, the dependent artifacts are notified and subsequently proper action can be taken.

The method involves three main components, i.e. (i) the requirement manager which responsible for managing requirements and for publishing change event messages to the event server, (ii) the event server which responsible for establishing traceability by handling initial subscriptions placed by dependent entities, and also listening for event notifications from the requirement manager(s) and forwarding event messages to relevant subscribers, and (iii) the subscriber manager which responsible for listening on behalf of the subscribers that it manages for event notifications forwarded by the event server.

EB assumes that the traceability links among artifacts have been established before event-based algorithms are run. Consequently, the algorithms are focused only to manage up-to-date traceability links based on changes that may occur during system operational time. The algorithms have been implemented in a tool prototype to manage and maintain traceability between requirements and UML artifacts as well as test cases.

2.4 Hypertext-Based Approach (HB)

Maletic et al. in [16, 17] propose an approach uses hypertext model that allow complex linking as well as versioning of links. Also, Sherba in [18, 19] propose a hypertext-based traceability relationships generation using open hypermedia and information integration.

The approach utilizes XML as the main tool for representing models and created links. The models and their links are converted into XML-based representation. Models are categorized into anchor model and target model. The links are established between anchor and target model with particular link types, i.e. causal, non-causal, and or navigational links. Once the model-to-model traceability links have been established, meta-differencing mechanism is used to indicate if some changes have been occurred in the models. The evolution is supported by a fine-grained versioning technique. The scope of tracing includes all types of artifacts.

2.5 Feature Model-Based Approach (FB)

Riebisch and Pashov in [20, 21] describe feature model-based method for requirement traceability. They utilize feature modeling which describes a requirements as an overview and models the variability of a product line. A feature model consists of a graph with features as nodes and feature relations as edges. If the number of features is very high, then the representation of features and their relations are displayed by tables. FB is applied for the definition of a product by a customer. Every feature describes a property of a product from the customer's point of view. There are three categories of features, i.e. (i) *functional features* (ii) *interface features,* and (iii) *parameter features.*

The features are structured by hierarchical relations. Classifications of feature relations are (i) *hierarchical relations* which describe the sequence of decisions of products. The most important features are placed higher in the hierarchy, (ii) *refinement relations* which describe relations of generalization and specialization as well as aggregation, and (iii) *requires* or *excludes* relations or *multiplicity-grouping relations* which describe constraints between variable features that have an influence on the

sequence of decisions of products. The scope of traceability includes requirements to features and elements of the solution, i.e. object model and source code.

2.6 Value-Based Approach (VB)

Zemont in [22] proposes a framework for assessing the value that traceability can provide to an organization. Furthermore, Heindl and Biffl in [23] propose value-based requirement tracing. This approach provides technical support to perform requirements tracing as well as take value and cost considerations into account. Thus, it provides a technical model and an economic model for requirement tracing based on some criteria.

VB consists of five processes, i.e. (i) *requirements definition*, that is identifying atomic requirements and assigning an identifier to each of them, (ii) *requirements prioritization*, that is estimating the value, risk, and effort of each requirement, (iii) *requirements packaging*, that is identifying clusters of requirements, (iv) *requirements linking*, that is establishing traceability links between requirements and other artifacts, and (v) *evaluation*, that is utilizing generated traces for certain purposes, e.g. to estimate the impact of change for particular requirements.

VB combines a manual and semi-automated way in obtaining the traceability links and in performing a change in the software artifacts.

2.7 Scenario-Based Approach (SB)

Egyed et al. in [24-26] propose scenario-based approach. SB uses a hypothesized trace information that have to be manually entered. Then, it uses runtime information to creating trace links. Test case scenarios are executed on a running system and execution information is obtained using a monitoring tool. The information is then combined with the hypothesized trace information to form a footprint graph. This graph shows the relationship among artifacts in the system.

The traceability links are created automatically, but the hypothesized trace information must be manually entered. In SB, traceability links can only be created once a running system is available.

3. Evaluation Framework

Buckley et al. in [27] propose the taxonomy of software evolution based on the characterizing mechanisms of change and the factors that influence these mechanisms. The taxonomy is organized into the following logical groupings: *temporal properties* that is a dimension of software evolution that captures timing aspect of change; *objects of change* that captures location aspect of change, i.e. which part of software can be changed; *system properties* that captures the characteristic of software while it is changed; and *change support* that captures support mechanism while software is being changed [27].

The temporal properties include *change frequency* which detects if software can be changed at continuously, periodically, or arbitrary intervals; *change history* which is supported by software versioning capability; *time of change* which categorizes software change capability into compile-time, load-time, or run-time change capability; and *anticipation* which refers to the time when the requirements for a software change are foreseen [27].

The objects of change include *artifact* which indicates what artifacts can be changed by the software; *granularity* which distinguishes level of granularity that can be changed into coarse, medium, and fine granularity; *impact* which indicates the impact of a change, i.e. local or system-wide impacted, and *change propagation* which denote if the software, upon a change, has capability to propagate to other part of artifacts. The propagation is implemented either by change propagation, change impact analysis, traceability analysis, or effort estimation features [27].

The system properties contain *availability* which indicates if the software has to be permanently available or not while a change is being made; *activeness* which indicates whether the software is reactive, i.e. changes are driven externally, or proactive, i.e. changes are driven by itself; *openness* which indicates whether the software is an open systems, i.e. built to allow for extensions, or closed system, i.e. do not provide a framework for extensions; and *safety* which distinguishes between static and dynamic safety [27].

The change support includes *degree of automation* which distinguishes between automated, partially automated, and manual change support; *degree of formality* which indicates whether the change support is implemented based on some underlying mathematical formalism or not; and *change type* which distinguishes between structural and semantic changes [27].

4. Evaluation of the Approaches

This section describes a comparative evaluation of various traceability approaches. The evaluation focus is primarily on their capability to support software evolution. The initial results obtained from this evaluation can be used to indicate to which extent an approach satisfies some features in term of its support for software evolution.

4.1 Temporal Properties

IR approach generates traceability link by processing software artifacts in the form of textual file format. Any other file formats will be transformed into textual file format before being processed in the preprocessing stage. If a change is to be performed in one of the software artifacts, then the changed artifact has to be first transformed into

textual file format, as well as other impacted artifacts. The next step is then recovering the traceability links into the new version of software traceability based on the change. Traceability generation as well as traceability recovery in IR approach does not directly invoke into the executable elements of the software. Thus, any changes in software artifacts have to be **recompiled** to get new version. IR approach implements automatic and dynamic retrieval methods. It means that traceability can be automatically generated for the first time as well as be automatically and dynamically recovered or updated when a change has been applied on particular artifact. It does not depend upon the existence of pre-established links. This characteristic allows IR approach to provide **versioning** mechanism. IR approach generates as well as recovers software traceability links in **arbitrary** period, depending on a change that can be performed in arbitrary period also.

RB approach generates traceability link by processing software artifacts in the form of either XML-based or DOM-based file format. Like IR approach, RB approach will transform any other file format into XML or DOM file format before it can be further processed by the approach. RB approach also does not directly invoke into executable element of the software when applying the rules, since the rules are only invoked into RSD, UCD, and AOM. If a change is implemented to a particular artifact, then RB approach will regenerate traceability links the way it generate links for the first time. If that change is propagate into the source code then it has to be **recompiled** to get the new software version. **Versioning** in RB approach is possible since the links regeneration is independent from the previous established links. RB approach also allows **arbitrary** changes applied on the software since changes can be performed in arbitrary mode as well.

EB approach assumes that traceability links among artifacts already exist in the maintained system and therefore it has to maintain those links during operational system periods. When a change is happened to the requirements, the dependent artifacts will be notified and some proper actions will be taken in order to update the traceability links into the new version. When updating the links, EB does not directly invoke into the executable elements of the software. Thus, like previous approaches, if a change in requirement propagates into its source code, then the software must be **recompiled** to make a new version. EB approach is run under distributed environment framework allowing more than one users send a change proposal at the same time. Changes are then resolved and recorded in the event log in each artifact in parallel way. These event logs allow EB approach to have **versioning** feature as well as **parallel** mode for applied changes. The changes can be applied on timely fashioned manner, thus in **arbitrary** mode.

HB approach applies traceability links creation for all types of non-executable artifacts. This means that this approach does not directly invoke into the executable elements of the software. Thus, any changes in software artifacts have to be **recompiled** to get new version. HB approach utilizes meta-differencing mechanism allowing system to indicate if some changes have been occurred in the models. This is a fine-grained **versioning** technique that the approach supports for evolution. Like other previous approaches, changes that have to be performed can be done in **arbitrary** moments.

FB approach does not directly invoke the executable elements when performing a change. If a change has to be performed in source code, then the source code has to be **recompiled** to get new version. In FB approach, traceability links are built with a particular feature management tool. This tool usually saves the built links in particular file format. It allows multiple **version** of document that is saved in different time. If any changes have to be performed in the impacted artifacts, they can be done in **arbitrary** mode.

VB approach generates traceability links in three different levels of detail, in manual way by human investigators. These links are applied on non-executable artifacts. In the evaluation stage, the available traces are utilized to estimate the impact of change. Any changes in software artifacts have to be **recompiled** to get new version. VB approach has no versioning feature since the links are built in manual way. However, changes that have to be performed can be done in **arbitrary** moments.

In fact, SB approach generates link from runtime execution of software system. But, if a change in requirement propagates into its source code, then the software must be **recompiled** to make a new version. Versioning in SB approach is impossible since traceability links can only be created once a running system is available. Like other previous approaches, changes that have to be performed can be done in **arbitrary** moments.

However, none of the approaches have capability to anticipate the change of the requirements in the future. Based on the above explanation, Table 1 below shows the evaluation result on the temporal properties aspect.

Table 1. The Evaluation on 'Temporal Properties' Dimension

Aspects	IRa	RB	EB	HB	FB	VB	SB
Time of Change							
• Compile	√	√	√	√	√	√	√
• Load							
• Runtime							
Change History							
• Versioning	√	√	√	√	√		
• Sequential							
• Parallel			√				
Change Freq.							
• Continuous							
• Periodic							
• Arbitrary	√	√	√	√	√	√	√

4.2 Object of Change

IR approach generates traceability links between free text documentations and source code [2, 7], between high-

level and low-level requirements documents [4], and between requirements documents and UML artifacts, source code and test cases [3, 8, 9, 28]. Therefore the artifacts scope of IR approach includes all **high-level** and **low-level** artifacts. This approach has a **fine-grained level** of granularity since links can be created at a method level in the source code. Any changes that are applied to the system may affect only to the other part of the same artifact. For example, a change in the local variable name of particular class may affect only to the class belonging the variable. But in other case, a change in any artifacts may affect to other artifacts in other level, for example, a change in the requirement may affect to the source code, design document, and test cases. Thus, the impact of change is **local** as well as **system-wide**. IR approach provides **traceability analysis** to support change propagation process in the context of evolution.

RB approach generates traceability links between RSD, UCD, and AOM [10, 11], and among all textual artifacts as long as those artifacts are in DOM trees format [12]. The artifacts scope of RB approach includes all **high-level** and **low-level** artifacts. This approach has a **fine-grained level** of granularity since links can be created at a method level in the source code (in the 'xlinkit' tool). Using the same reason that is mentioned in IR approach, this approach can make **local** as well as **system-wide** impact upon a change has been implemented. RB approach also provides **traceability analysis** to support change propagation process.

EB approach assumes that there are already available links between requirements and UML artifacts including test cases. Source code is excluded. The approach then runs its maintenance mechanism upon those artifacts and their links. The artifacts scope of EB approach includes all **high-level** and **low-level** artifacts (UML design documents, i.e. class diagram, sequence diagram, and other design diagrams are classified into low-level artifacts). This approach has a **medium level** of granularity since the source code is excluded from the established links. In EB approach, a change is only applied on requirements. The change is then used to analyze which parts of other artifacts are impacted. This change impacts on **system-wide** level. EB approach provides **traceability analysis** as well as **change impact analysis** to support software evolution.

HB approach generates traceability links among all types of artifacts, including source code. Thus, the artifacts scope of HB approach includes **high-level** and **low-level** artifacts. This approach has a **fine-grained level** of granularity since links can be created at a method level in the source code. A change can be applied on any types of artifacts and the impact of change can be in **local** or **system-wide** level. HB approach provides **traceability analysis** to support software evolution.

FB approach generates traceability links among requirements, object model, and source code. The artifacts scope of FB approach includes **high-level** and **low-level** artifacts. This approach has a **medium level** of granularity since it does not actually invoke into the source code, instead it only concerns for the packages bundling the source code. A change can be applied on any types of artifacts and the impact of change can be in **local** or **system-wide** level. FB approach provides **traceability analysis** to support change propagation process.

In the case study performed by [23] to demonstrate VB approach, the traceability links are manually generated between requirements documents and design elements, and between requirements documents and source codes. Thus, the artifacts scope of VB approach includes **high-level** and **low-level** artifacts. This approach has a **fine-grained level** of granularity since links can be created at a method level in the source code, although it is done manually. In VB approach, a change is possible to occur only in requirements documents allowing the change impact into **system-wide** level. VB approach provides **traceability analysis** as well as **change impact analysis** to support software evolution.

SB approach generates traceability links among requirements, design model, and source code. The artifacts scope of SB approach includes **high-level** and **low-level** artifacts. This approach has a **fine-grained level** of granularity since links can be created at a method level in the source code. A change can be applied on any types of artifacts and the impact of change can be in **local** or **system-wide** level. SB approach provides **traceability analysis** to support software evolution.

Based on the above explanation , the evaluation result of the 'object of change' aspects can be seen in Table 2 below.

Table 2. The Evaluation on 'Object of Change' Dimension

Aspects	IRa	RB	EB	HB	FB	VB	SB
Artifact							
• High-level	√	√	√	√	√	√	√
• Low-level	√	√	√	√	√	√	√
Granularity							
• Coarse							
• Medium			√		√		
• Fine	√	√		√		√	√
Impact							
• Local	√	√		√	√		√
• System-wide	√	√	√	√	√	√	√
Ch Propagation*							
• CIA			√			√	
• TA	√	√	√	√	√	√	√
• EE							

* CIA: change impact analysis; TA: traceability analysis; EE: effort estimation

4.3 System Properties

Requirement traceability approaches that are mentioned in the previous section work in similar manner. They first create traceability links among artifacts (for those which assume that the links are not available before the systems are run), then use the established links to maintain the software systems, i.e. to estimate the impact of a change on a particular artifact, and when a change occurs

in one of the artifacts, they recover the traceability links based on that change.

When software artifacts are being traced or those artifacts' links are being recovered, the software itself does not have to be at run-time mode. After a change has been implemented on the impacted artifacts, and subsequently the change is propagated down into the source code element, then the software need to be recompiled to get a new version. This situation complies with all of the approaches, even if the approach is run under the distributed environment framework, such as EB approach. Therefore all of the approaches are **partially available**, that is, the running software systems can be stopped while it is being modified. In other word, the approaches support for only partially availability of software systems.

Furthermore, all of the approaches are **reactive** since changes on the artifacts must be driven by an external agent, i.e. user or stakeholder. All of the approaches support for **openness** of a software system as they facilitate the software to be extended or modified. If a particular requirement is changed, then the approaches can either regenerate links or analyze the available links, to indicate which other parts of the software will be impacted and therefore should be modified.

Also, all of the approaches support for **static safety** feature as the tracing processes guarantee a certain degree of behavioral safety, in which the change is behavior-preserving with respect to the original behavior (although there is no formal proof of this).

Based on the above explanation , the evaluation result of the 'system properties' aspects can be seen in Table 3.

Table 3. The Evaluation on 'System Properties' Dimension

Aspects	IRa	RB	EB	HB	FB	VB	SB
Availability							
• Partially avail.	√	√	√	√	√	√	√
• Permanently av.							
Activeness							
• Reactive	√	√	√	√	√	√	√
• Proactive							
Openness							
• Open	√	√	√	√	√	√	√
• Closed							
Safety							
• Static	√	√	√	√	√	√	√
• Dynamic							

4.4 Change Support

IR, RB, and HB approaches generate traceability links (and recover the links based on a change) in an **automated** way. EB generates requirements changes in an **automated** way. VB, FB, and SB are **semi-automated** since they combine a manual and automated way in obtaining the traceability links and in performing a change in the software artifacts.

All of the approaches are implemented on some underlying mathematical model especially for the core tracing algorithms while some other parts are based on informal model. Hence, all of them are **semi-formal**. The informal model can also be applied when user have to decide whether the result are accepted or rejected, since no mathematical model can be used to judge the decision. It is based on intuition and knowledge of the user.

All of the approaches support for **structural** and **semantic** change. The rationale is as follows. All of the approaches are for requirement tracing whether it is performed for the first time (establishment) or after the traceability has been accomplished (recovery due to a change). Therefore, a change can be made on one of the artifacts, which can be at requirement documents, design documents, or source code. All of the artifacts are stored as files. So, the change is made on one of those files and it can be any kind of change, i.e. addition, deletion, or modification. It can be structural, semantic-preserving, or semantic-changing change. Table 4 shows the results, based on the above explanation.

Table 4. The Evaluation on 'Change Support' Dimension

Aspects	IRa	RB	EB	HB	FB	VB	SB
Deg. of Automation							
• Automated	√	√	√	√			
• Semi-automated					√	√	√
• Manual							
Degree of Formality							
• Ad-hoc							
• Semi-formal	√	√	√	√	√	√	√
• Formal							
Change Type							
• Structural	√	√	√	√	√	√	√
• Semantic	√	√	√	√	√	√	√

5. Discussion

In the context of software evolution support, we can divide the discussion into two disjoint groups. The first is related to the similarity characteristics belong to each approach, and the second is the opposite.

The similarities arise among them is caused by the natural or inherent characteristics of requirement traceability process. For example compile-time change, arbitrary change, reactive, semi-formal, etc. These characteristics seem too hard to be improved, since the user intervention, i.e. the need for user justification in the final results, can not be avoided in the process.

The second group is related to the characteristics in which each approach have different value. The way to improve the value to achieve a better support for software evolution is by modifying the approach itself (if the algorithms enable) or by combining a particular approach to other approach that have a better value (if the technology is enable). For example RB approach may be combined with HB to achieve recovery capability as well as requirement semantic meaning capability (even though it needs further exploration from both of them).

6. Threats to Validity

First, the sources that are used to evaluate the approaches are mainly from the published research papers, especially from the international journals and or the conference proceedings. The papers usually contain brief and compressed information (due to space restriction) in that some other information probably were disappeared related to the long version one, i.e. dissertation report, or technical report. Therefore the justifications are made from the concise information. However, those papers were already published and well accepted in international community, in term of their information 'completeness' and clarity.

Second, the justifications are performed without any formal methodology. We use our comprehension from reading the papers and concluding the result based on our understanding and intuition. However, the initial result presented in the evaluation can be very useful to perform further and deeper evaluation of the approaches for future improvement, and also to welcome any open discussions.

7. Conclusion

In this paper, we have presented the evaluation of state-of-the-art requirements traceability approaches, especially in the context of software evolution. We have evaluated the approaches using the taxonomy of software evolution framework. The results showed us that so far, there is no approaches fully satisfied all of the requirements of traceability related capabilities that have to be accomplished to support software evolution. This means that much work have to be done to achieve the better approaches in the future.

Acknowledgements

The authors would like to thank Ministry of Higher Education (MOHE) Malaysia, Universiti Teknologi Malaysia (UTM), and Islamic Development Bank (IDB) for their financial support.

References

[1] O. Gotel and A. Finkelstein, "An Analysis of the Requirements Traceability Problem," in *Proceeding of 1st International Conf. on Requirement Engineering*, 1994.

[2] G. Antoniol, G. Canfora, G. Casazza, G. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, October, 2002, pp. 970-83.

[3] J. Cleland-Huang, R. Settimi, C. Duan, and X. Zou, "Utilizing Supporting Evidence to Improve Dynamic Requirements Traceability," in *Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE'05).* 2005.

[4] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Improving After-the-fact Tracing and Mapping: Supporting Software Quality Predictions," IEEE Software, 2005.

[5] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: The study of methods," *IEEE Transactions on Software Engineering*, vol. 32, no. 1, January, 2006.

[6] J. Lin, C. C. Lin, J. Cleland-Huang, R. Settimi, J. Amaya, G. Bedford, B. Berenbach, O. Ben Khadra, C. Duan, and X. Zou, "Poirot: A Distributed Tool Supporting Enterprise-Wide Automated Traceability," in *14th IEEE International Requirement Engineering Conference (RE'06)*, 2006.

[7] A. Marcus and I. J. Maletic, "Recovering Documentation-to-Source Code Traceability Link using Latent Semantic Indexing," in *Proceedings of the 25th IEEE International Conference on Software Engineering*, 2003.

[8] R. Settimi, J. Cleland-Huang, O. B. Khadra, J. Mody, W. Lukasik, and C. DePalma, "Supporting software evolution through dynamically retrieving traces to UML artifacts," in *Proceedings - 7th International Workshop on Principles of Software Evolution, IWPSE 2004 (In Conjunction with RE 2004)*, 2004.

[9] X. Zou, R. Settimi, and J. Cleland-Huang, "Phrasing in Dynamic Requirements Trace Retrieval," in *Proceeding of 30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, 2006.

[10] G. Spanoudakis, "Plausible and Adaptive Requirement Traceability Structures," in *Proc. 14th Int'l Conf. Software Eng. and Knowledge Eng*, 2002.

[11] G. Spanoudakis, A. Zisman, E. Perez-Minana, and P. Krause, "Rule-Based Generation of Requirements Traceability Relations," *Journal of Systems and Software*, 2004, pp. 105-27.

[12] C. Nentwich, L. Capra, W. Emmerich, and A. Finkelstein, "xlinkit: A Consistency Checking and Smart Link Generation Services," *ACM Transactions on Internet Technology*, vol. 2, no. 2, 2002, pp. 151-85.

[13] J. Cleland-Huang, C. K. Chang, S. Gaurav, J. Kumar, H. Haijian, and X. Jinchun, "Automating Speculative Queries through Event-based Requirements Traceability," in *Proceedings of the IEEE Joint International Conference on Requirements Engineering*, 2002.

[14] J. Cleland-Huang, C. K. Chang, and G. Y., "Supporting Event Based Traceability through High-

Level Recognition of Change Events," in *IEEE Proc. Int'l Computer Software and Applications Conf. (COMPSAC)*, 2002.

[15] J. Cleland-Huang, C. K. Chang, and M. Christensen, "Event-based traceability for managing evolutionary change," *IEEE Trans. on Software Engineering*, vol. 29, no. 9, Sept, 2003, pp. 796-810.

[16] J. I. Maletic, E. Munson, A. Marcus, and T. Nguyen, "Using a Hypertext Model for Traceability Link Conformance Analysis," in *Proceedings of 2nd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE'03)*, 2003.

[17] J. I. Maletic, M. L. Collard, and B. Simoes, "An XML Based Approach to Support the Evolution of Model-to-Model Traceability Links," in *Proceedings of 4th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE'05)*, 2005.

[18] S. A. Sherba, "Towards Automating Traceability: An Incremental and Scalable Approach," PhD Thesis, Department of Computer Science, University of Colorado, 2005

[19] S. A. Sherba, K. M. Anderson, and M. Faisal, "A Framework for Mapping Traceability Relationships," in *2nd International Workshop on Traceability in Emerging Forms of Software Engineering. (TEFSE '2003)*, Montreal, Canada 2003.

[20] M. Riebisch, "Supporting evolutionary development by feature models and traceability links," in *Proceedings - 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems*, 2004.

[21] I. Pashov and M. Riebisch, "Using feature modeling for program comprehension and software architecture recovery," in *Proc. of 11th IEEE Int'l Conf. and Workshop on the Engineering of Computer-Based Systems*, 2004.

[22] G. Zemont, "Towards Value-Based Requirements Traceability," PhD Thesis, Department of Computer Science, De Paul University, 2005

[23] M. Heindl and S. Biffl, "A Case Study on Value-Based Requirement Tracing," in *International Conference on Empirical Software Engineering (ESEC-FSE'05)*, Lisbon, Portugal, 2005.

[24] A. Egyed, "A Scenario-Driven Approach to Traceability," in *23rd International Conference on Software Engineering*, Toronto, Ontario, Canada, 2001.

[25] A. Egyed and P. Grunbacher, "Automating Requirements Traceability: Beyond the Record & Replay Paradigm," in *17th IEEE International Conference on Automated Software Engineering (ASE'02)*, 2002.

[26] A. Egyed and P. Grunbacher, "Supporting Software Understanding with Automated Requirements Traceability," *International Journal of Software Engineering and Knowledge Engineering*, vol. 15, 2005.

[27] J. Buckley, T. Mens, M. Zenger, A. Rashid, and G. Kniesel, "Towards a Taxonomy of Software Change," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 17, no. 5, 2003, pp. 309 - 32.

[28] X. Zou, C. Duan, R. Settimi, and J. Cleland-Huang, "Poirot:TraceMaker: A Tool for Dynamically Retrieving Traceability Links," vol. 2006, 2005.